APPENDIX A

Please Do not Print Appendix V-S 3/25/04

S O P C B U I L D E R
Patent Disclosure
April 20, 2001

Introduction:
This disclosure describes a set of software programs which include:

1) A graphical user interface (wizards)
2) A logic-generator
3) A logic-synthesis tool
4) A place-and-route tool
5) A programmable-logic configuration method

Items (1) and (2) contain novel elements, and are the subject of th is disclosure. In the preferred embodiment, item (3) is a standard preexisting logic-synthesis tool (e.g. Leonardo Spectrum, made by t he Exemplar corporation). Items (4) and (5) are generally provided by the programmamble logic device vendor. In the preferred embodiment, items (4) and (5) are facilities in the Quartus tool, made by the Altera corporation.

This set of software programs allows a user to describe a custom microprocessor system, including the CPU and peripherals, by operat ing control elements within the grapical user interface (1). After the description is complete, the user can realize a physical implementation of this custom system on a programmable logic device (e.g. an Altera Apex device).

Prior Art:

Microprocessor-based systems have existed since at least 1970 (Hoff, Faggin, et. al [intel 4004]), and have found widespread application s in a staggering variety of useful products. For many years, microprocessor systems were exclusively "Systems-on-a-board:" The C PU, memory, and each peripheral function each resided within their own integrated circuit (chip). These chips were connected together to form a working system by the traces of a printed-circuit board on which they were mounted. Many of today's microprocessor systems differ only in detail from this original arrangement. For example: since 1970, personal computers have achieved ubiquity, and today's

typical PC may be described as separate CPU, memory, and peripheral chips mounted on, and connected by, a printed-circuit board.

In many systems, however, there are advantages to combining CPU, peripheral, and/or memory functions on a single integrated circuit (chip). This higher level of integration (more function on a singl e chip) confers numerous advantages, including: Compact size, low per-unit cost, higher performance, ruggedness, and low power consumption. The advantages of implementing an entire microprocess or system on a single chip is well-understood. In recent years, many commercial tools and products have evolved specifically to aid in t he design, implementation, and commercial application of SOC (System-On-a-Chip) products.

In the prior art, there are two common methods for implementing integrated microprocessor systems:

1) Standard "off-the-shelf" microcontroller components.
2) Customized microprocessor systems implemented on an ASIC.

**** Microcontroller-based systems:

Microcontroller components include a CPU, memory, peripherals, and I/O components integrated onto a single silicon die. Microcontrollers are sold as standard components. Microcontroller vendors typically offe r a fairly large number of product "variants". Each variant has a different particular mix of peripherals, memory types and sizes, and/or I/O resources.

At present, microcontrllers are available in a wide variety of pric e, performance, and peripheral-integration options. For example, The Microchip corporation of Chandler, Arizona sells a simple 8-bit microcontroller with memory and I/O peripherals in an 8-pin package for under $0.50 (PIC-12XXX). Motorola (of ??, Illinois) sells a li ne of "PowerQuic" microcontrollers to the networking and communication s market. These devices include a powerful 32-bit CPU, rich peripher al content, and specialized sub-units which can perform networking and

communications functions. Other microcontroller vendors include Intel, Zilog, Hitachi, Philips, ST Microelectronics, and many other s.

System designers typically select the standard microcontroller component which most closely meets their needs. The electronic content of some embedded microprocessor systems can be implemented entirely within a standard microcontroller component (e.g. microwav e ovens, television remote-controls, etc). More complicated systems, however, generally require some system-specific logic in addition t o the peripherals and memory available on a standard microcontroller. Most high-function microcontroller-based embedded systems generally include one or more chips in addition to the standard microcontroll er component.

**** ASIC-based systems:

Microcontroller components allow system designers to take advantage of integration by offering standard "pre-packaged" collections of CPU, peripheral, and memory functions. This allows many of the chips in a traditional "system-on-a-board" to be combined into a single chip. But, in some applications, cost, size, and/or power-consumption requirements may require an even higher level of integration. In m any systems today, all of the system logic, including system-specific functions which are not offered in standard microcontroller product s, are included on a single chip. System designers who require this level of integration generally need to create a chip which is custom-designed to include all the required components--often including a CPU, peripherals, and memory. Chips which are custom-designed to suit a particular application are often referred to as ASICs (Application-Specific Integrated Circuts). The process of designing, verifying, fabricating, and testing ASICs (especially on es complicated-enough to include a CPU and peripherals) is understood to be a time-consuming, risky, and punishingly expensive undertaking. Today, the design and verification process for a complex ASIC typically takes anywhere from 9 months to 2 years, requiring a team of engineers using a variety of software tools which often cost $100,0

00
or more (e.g. Synopsys DC, Formality, Avant!, etc.). The tooling charges for a state-of-the-art ASIC can cost hundreds of thousands of
dollars. It is common for companies to invest more than a year and more than a million dollars in the design of a single ASIC. And th is
level of investment does not guarantee a working result. A nonnegligible percentage (~20%?) of ASICs are not functional on fir st
silicon, and require yet-more engineering effort, time, and money t o
bring to production.

Thus, the level of SOC-integration allowed by an ASIC has traditionally only been available to system designers with deep pockets, vast engineering resources, and an apetite for risk. Stil l,
the need for integration in many systems is so compelling that it i s
worth the cost, time, and risk of an ASIC-design process to achieve a
system on a single chip.

**** Programmable Logic.

For more than a decade, programmable logic has offered an alternati ve
to ASIC's high NRE costs and lengthy development schedules. Using programmable logic chips (from vendors such as Altera, Xilinx, and Actel) designers can realize a custom logic function in a single chip. A typical engineering and verification schedule for a moderately-complex programmable logic design ranges between a few d ays
and three months. Most modern programmable logic chips can be reconfigured with a new design essentially instantly, and can be updated with new designs at-will. Chip-specific NRE costs for a programmable logic design (not including engineering) is typically less than $10,000. Thus, programmable logic devices offer system designers access to a high level of custom integration at a fractio n
of ASIC cost and time, with very low risk (design errors can be fix ed
in-place).

Historically, programmable-logic devices have suffered from the following limitations (relative to ASICs):

1) Substantially high per-unit cost.
2) Lower performance (clock frequency).
3) Lower density (limited complexity on a single chip).

**** Summary: Implementation Choices

In the prior art, system designers have used microcontrollers, ASICs, and programmable logic according to the following (very rough) de-facto guidelines:

* Standard microcontrollers are the solution-of-choice for systems with little or no custom hardware (logic) content. This is true, for example, when most system functions are implemented in software.

* ASICs are used in systems where per-unit cost, performance, or power-consumption are critical. Such applications often include high-volume and/or portable devices.

* Programmable logic is used in systems where flexibility and rapid time-to-market are critical, and which can tolerate a high per-unit cost. Such applications have included network infrastructure and communications equipment.

**** Economic Backgroung of the Current Invention:

Until recently (CY1999 and later) programmable-logic devices have not offered high-enough density to integrate CPU-based systems. A CPU core by itself --without bus logic, peripherals, or memory-- would have occupied half or more of the logic in a high-density PLD, and cost more than ten times as much as the corresponding standard microcontroller component. Examples of high-density PLD devices (circa 1998) include Altera 10K100E and [...fill in...]. These devices were not large enough (not enough logic on a single chip) to implement cost-effective microprocessor-based systems.

Recent developments in Silicon wafer fabrication, logic design, synthesis, and place-and-route technology have tended to reduce the cost, performance, and density limitations of programmable logic.

These technological advances can be broadly characterized as the natural continuation of "Moore's law" over the past three years. These developments have caused programmable logic devices (PLDs) to increase in density, performance, and cost-effectiveness every year.

Economies-of-scale in selling a general chip to many customers, as opposed to a specific chip for one application, have tended to "narrow the gap" in density, performance, and cost between PLDs and ASIC solutions.

This invention was developed just as PLDs were reaching the market with sufficient density to implement a 32-bit RISC CPU, a useful set of peripherals (UART, timer, memory interfaces, etc), and still leave enough logic to implement custom (user-defined) functions.

**** Other Microprocessor Cores

A Microprocessor "core" is an abstract logic description (e.g. HDL code) of the logic which implements a microprocessor function. Microprocessor cores are well-known in the prior art. Designs for microprocessor cores can be licensed from several vendors including ARC, Tensilica, ARM, etc. A system designer can implement a properly-licensed microprocessor core from any of these vendors in either an ASIC or a PLD. The present invention includes a microprocessor core as one of its components, but the core by itself is not the novel element.

**** Other PLD/Processor combinations

The idea of combining a microprocessor function and an array of programmable logic elements on a single chip is known in the prior art. Chips which perform this function are available from the Triscend corporation (A7 and E5 devices, for example). These devices differ from the current invention because the CPU core itself is "hard" logic--it is implemented as a fixed logic function in dedicated hardware on the chip. In the current invention, the CPU core is "soft" logic--it is implemented as a particular configuration of undifferentiated programmable logic elements, with no hard function at all.

**** Novel Elements of the Current Invention

The current invention comprises a novel combination of elements, each of which is known individually in the prior art. Novelty arises from this particular useful combination of these elements. They are:

1) A graphical user interface which allows users to specify a complete embedded microprocessor system, including the arrangement of such elements as the CPU, peripherals, and memory interfaces (The GUI).

2) A program which generates an abstract description (HDL) of the logic which implements the specified system. (The generator-program).

3) A set of programs which convert this abstract logic description into a PLD-specific configuration file (Synthesis tool + Quartus).

4) A programmable logic device which can be configured to contain the specified system, perhaps in combination with other design elements, to implement the user-defined function.

This arrangement is novel. Graphical user interfaces for configuring hardware components like microprocessors are knwon in the prior art (ARC cores' "ARChitect"). Graphical user interfaces for specifying a collection of microprocessor peripherals are also known (Triscend's "FastChip" tool). But the particular application of such a tool to a soft-core microprocessor, resulting in a system which is entirely implemented as "soft" logic in a PLD, is novel.

**** The Preferred Embodiment

The remainder of this document describes the preferred embodiment of the current invention. It is meant to be instructive, but not limiting.

**** Overall Tool Structure

The combination of elements 1..4 above is collectively referred to

as
the "System-Builder" tool. In the preferred embodiment, the system-builder tool

**** The graphical User Interface.

In the current invention, it is desirable to guide the user through the system-specification process according to a step-by-step sequential procedure. At each step, the user answers a set of "questions" (i.e. enters information or makes selections using GUI control elements). These "questions" (choice of control elements) may
or may not be modified as a result of user interactions from previo us
steps.

The preferred embodiment uses the "wizard" GUI-paradigm to guide th e
user through a step-by-step sequence of choices. "Wizards" are well-known in the prior art, and appear in such ubiquitous software tools as Microsoft Office (Microsoft Corp, Redmond, WA). Wizards a re
sufficiently-common interfaces that there are commerically-availabl e
authoring tools which allow software designers to create wizard interfaces (xxxx??, also from Microsoft Corp). A flowchart diagra m
of a typical wizard-interface appears in Fig. 1.

Wizards generally consist of an ordered list of "pages." Each page presents the user with its "contents," which are a set of common graphical user interface control elements (check-boxes, radio butto n
groups, drop-down selections, etc.). Each page includes a standard group of wizard-navigation controls, usually implemented by four buttons at the bottom of the page. These controls allow the user t o
proceed through the flow ("Next"), back-up to a previous page ("Prev"), force the process to completion ("Finish"), or abort the wizard process altogether ("Cancel"). These choices are usually offered on every page of the wizard. The preferred embodiment uses a
standard set of wizard-navigation controls in a conventional layout similar to that found in (for example) the aforementioned Microsoft products and in other Altera "MegaWizard" user-interfaces. The box in Fig. 1 labeled "Generate Result" will implement a different action, potentially a very compilcated one, which depends on the application at hand. Almost always, this will result in the

generation or modification of files on the user's computer (or network). The "letter" wizard in Microsoft Word, for example, would create a Word document in the format of a letter, with various contents and format specified by the user during the wizard-input process.

Figure 2 shows details of the "Generate Result" step for the Wizard in the current invention. This wizard is known as the "System-Builder" Wizard (herein after: SBW). The SBW aggregates all of the user-input choices from the various preceding wizard pages and saves them to a file. In the preferred embodiment, these data are saved in the ad-hoc "PTF" file-format, which is a semi-human-readable, editable ASCII format. "PTF" is a specific case of the general class of ASCII data-formats well-known in the prior art. The choice of the PTF-format in specific, and ASCII save-formats in general, is instructive but not limiting--any of a variety of well-known save-file formats would work equally well. An example of the PTF file format appears in Appendix A.

In the preferred embodiment, the generator program is a "command-line" Perl-script. This means that the program can be run by "typing a command" at a shell-prompt, that it takes files as input, and that it produces text-only output and status information while it runs (in other words, it does not have its own GUI). The System Builder wizard "runs" the generator program "as if it had been run from the command line." Methods for one program to run or launch another sub-program are well-known to those of ordinary skill in the art. Before the System Builder wizard launches the generator program, it displays a console window. Any status- or progress messages produced by the generator program are displayed in this console window. This allows the user to monitor the progress of the generation process.

In the preferred embodiment, The "generator program" in Fig. 2 is a Perl script which, ultimately, produces an "EDF"-file. EDF is an industry-standard netlist format for describing logic circuits at a very low (usually implementation-specific) level. The EDF format is known to those of ordinary skill in the art. In the preferred

embodiment, the final product of the System Builder wizard is an EDF file which describes a logic circuit which implements the embedded processor system. This is the embedded processor system which the user described using the GUI-controls in the System Builder wizard pages (and which, equivalently, is described by the contents of the system PTF-file). This EDF-file is referred to as the "system netlist." In the preferred embodiment, the EDF file is a "synthesized netlist", which means that it contains an exhaustive list of every technology-specific logic element (LE) in the design, and complete information about how they are wired-together. This EDF-file is one of the final products of the generator program. This EDF-file can be read into Quartus and used as a component (sub-module) in a user's logic design.

The EDF-file produced by the generator-program implements an entity known as the "system module." The system module is a block of logic which impements the entire embedded processor system specified during the wizard-flow process. The internal structure of this system module is shown in Fig. 4. (Note that this structure represents the structure used to -specify- the system module, but that this structure may have been lost if the design is, for example, "flattened" during the synthesis process. Thus, Fig. 4 shows "all the ingredients that went into the cake," even though they may no longer be distinct in the final EDF-output) The system module contains an instance of every module specified in the system's PTF-file, and additional modules which are created ab-initio by the generator program. One of the "created" modules is the system-module itself, which is a composite module that instantiates and connects all other sub-modules in the system. Another "created" module is the PBM (Peripheral Bus Module). This module contains all the logic and wiring necessary to connect the master-module to the interface ports on all the slave modules. The PBM may contain (but is not limited to) the elements shown in Fig. 4b. In the preferred embodiment, this includes:

* Address-decoding
* Wait-state generation

* Databus multiplexing
* Interrupt-control and prioritization.

All of these elements are familiar to board- and chip-level embedded systems designers, and are familiar to those of ordinary skill in the art.

"Perl" is a general-purpose, open-source, freely-available interpreted programming language. Installable versions of Perl, and extensive information about same, can be obtained from www.perl.org. Perl is commonly used for system-task scripting, text-file translation and manipulation, and web automation.

Fig. 3 shows details of the generator program's operation in the preferred embodiment. First, the generator program reads the PTF-file which was created in step 1 of Fig. 2. This PTF-file contains a description of all "modules" in the user-specified system. A "module" is either a master (e.g. CPU) or slave (e.g. UART, timer, I/O block) in the user-specified system. The generator program parses the PTF-contents and extracts a list of all modules in the system. Each module must be an instance of a predefined library-component type. Library components are stored in a directory tree (by default, in the preferred embodiment, in the directory c:/Altera/Excalibur/sopc_builder/components). Each library component definition consists of a directory which contans a PTF-file and any other files (e.g. HDL, C-software and/or Perl-scripts) which are specific to that component. Each module's description in the system PTF-file gives the name of its associated library component. When the generator program runs, it attempts to find the library component associated with each module in the system. It does so by checking a search path given by (for example) a registry key or environment variable. Library search-paths, registry keys, and environment variables are known to those of ordinary skill in the art. If the generator program is unable to find the library component associated with a module in the system, then the generator program terminates with an error.

Each library component may (optionally) have its own "private" generator program which (for example) may produce an HDL description (e.g. Verilog or VHDL) of the particular instance (module) based on the description of that module in the system-PTF file. Alternatively, simple library components may contain preexisting HDL files--with no need for "on the fly" generator scripts. For each module in the system, the master (system) generator-program will run the associated library-component generator-program, if it exists. (information about a library components' generator program, or lack thereof, is given in the library-comonent's definition).

Each library-component's generator program will have two results (in general--there are no specific limits on what other related or unrelated actions taken by a component generator program). First, the "sub-generator" will probably produce an HDL file which implements the module-in-question. Second, the sub-generator may add information to the system's PTF-file about the module which was just generated. This additional information may include (but is not limited to) a list of HDL-files which were created and a list of (and information about) each I/O port on the module which was created.

After each module is generated, the master generator program extracts a list of synthesizable HDL-files assocated with that module (this information is contained in the system's PTF file). This process (library-search and sub-generator-program run) is repeated for each module described in the system PTF file.

At the conclusion of this process, an implementation (e.g. synthesizable HDL file) will have been created for every module in the specified system). The master generator script must then generate the PBM (Peripheral Bus Module) and the System Module. The

System Module contains no logic per-se, and is merely a "container" which instantiates and connects all of the other system components.

In the preferred embodiment, the various master and slave modules, the PBM, and the system module are all implemented (generated) as one or more (each) synthesizable Verilog files. The master generator program automatically synthesizes these files as a group ("project") in order to create a single EDF implementation of the system module and its contents. The process of logic synthesis (convering an HDL logic description into a technology-specific netlist) is well-known to those of ordinary skill in the art. Several vendord produce commercially-available synthesis tools, including Leonardo Spectrum, made by the Mentor Graphics Corporation, Synplify, made by the Synplicity Corporation, and FPGA Express, made by the Synopsys Corporation. In the preferred embodiment, the master generator program creates a command-file for Leonardo Spectrum. This command file provides instructions, including a list of HDL files, for Leonardo Spectrum to synthesize the project and produce an EDF file.

In the preferred embodiment, a version of Leonardo Spectrum is "bundled with" (distributed and installed with) the generator program and the system builder wizard. The master generator program runs (launches) Leonardo Spectrum in such a way that it reads the command-file and synthesizes the project. The result will be the system module's EDF netlist.

Figure 5 shows a detailed flow of the "Generate PBM" step shown in Figure 3. All of the steps in figure 5 are executed as part of the master generator program's normal operation. First, the master generator program extracts system-connection information from each module defined in the PTF-file (this information is contained in the SYSTEM_BUILDER_INFO sections visible within each MODULE section in Appendix A). In addition, the master generator program extracts a list of all I/O ports on each module (this information is contained in the PORT_WIRING sections visible within each MODULE section in Appendix A). For each port on each module, the master generator program (MGP) determines if the port is used to interface to the master (via the PBM), or if the port serves some unrelated purpose. Ports which are not used for the PBM-interface are "promoted," and appear as (and are connected to) a top-level I/O port on the final

System Module. The master generator program keeps a list of all module-ports which are used to connect to the PBM. Information about how the port is used (including whether or not it is used to connect to the PBM) is contained in each port's PORT section within the PORT_WIRING sections, as shown in the example in Appendix A. PORT sections also contain information about how the port connects to one of the several internal PBM functional units shown in Fig. 4b. The master generator program can use this information to generate logic for each of the functional untis contained within the PBM and, of course, the PMB itself.

A P P E N D I X A:

Example PTF file-format

```
**************************************************************
****************          BEGIN EXAMPLE          ****************
**************************************************************
#
# file: ./ref_32_system.ptf
# date: 2001.03.13 18:45:06
# generated by a perl script
#
  SYSTEM ref_32_system
  {
    WIZARD_SCRIPT_ARGUMENTS
    {
      mainmem_module = "ext_ram";
      datamem_module = "ext_ram";
      maincomm_module = "uart1";
      gdbcomm_module = "uart1";
      germs_monitor_id = "ref32-v1.1";
      reset_offset = "0x0";
      reset_module = "boot_monitor_rom";
      vecbase_offset = "0x0";
      vecbase_module = "ext_ram";
      Principal_Tri_State_Data_Bus = "ext_ram_bus";
      skip_synth = "0";
      device_family = "APEX20KE";
      clock_freq = "33333000";
      hdl_language = "verilog";
      compiler = "quartus";
    }
```

```
MODULE ref_32_system_cpu
{
   class = "altera_nios";
   class_version = "1.1";
   HDL_INFO
   {
      Synthesis_HDL_Files = "./ref_32_system_cpu_dr.v,./ref_3
2_system_cpu_ar.v,./ref_32_system_cpu_cr.v,./ref_32_system_cpu_regi
ster_ram.v,./ref_32_system_cpu_major_opcode_table.v,./ref_32_system
_cpu_subtable_w.v,./ref_32_system_cpu_instruction_decoder.v,./ref_3
2_system_cpu_cpu_core.v";
      MIF_Files = "./ref_32_system_cpu_major_opcode_table.mif
,./ref_32_system_cpu_subtable_w.mif";
   }
   SYSTEM_BUILDER_INFO
   {
      Is_Bus_Master = "1";
      Is_Enabled = "1";
      Instantiate_In_System_Module = "1";
      Data_Width = "32";
      Address_Width = "21";
      Date_Modified = "2001.0.26.11:41:32";
   }
   WIZARD_SCRIPT_ARGUMENTS
   {
      num_regs = "256";
      shift_size = "7";
      mstep = "1";
      multiply = "0";
      wvalid_wr = "0";
      rom_decoder = "1";
      mainmem_module = "ext_ram";
      datamem_module = "ext_ram";
      maincomm_module = "uart1";
      gdbcomm_module = "uart1";
      germs_monitor_id = "ref32-v1.1";
      reset_offset = "0x0";
      reset_module = "boot_monitor_rom";
      vecbase_offset = "0x0";
      vecbase_module = "ext_ram";
   }
   PORT_WIRING
   {
      PORT mem_be_n
      {
         direction = "output";
         width = "4";
         avalon_role = "byteenablen";
```

```
}
PORT irq_number
{
   direction = "input";
   width = "6";
   avalon_role = "irqnumber";
}
PORT ifetch
{
   direction = "output";
   width = "1";
   avalon_role = "ifetch";
}
PORT clk
{
   direction = "input";
   width = "1";
   avalon_role = "clk";
}
PORT mem_addr
{
   direction = "output";
   width = "21";
   avalon_role = "address";
}
PORT mem_is_32_bits
{
   direction = "input";
   width = "1";
   avalon_role = "memis32bits";
}
PORT irq
{
   direction = "input";
   width = "1";
   avalon_role = "irq";
}
PORT data_from_cpu
{
   direction = "output";
   width = "32";
   avalon_role = "writedata";
}
PORT data_to_cpu
{
   direction = "input";
   width = "32";
   avalon_role = "readdata";
```

```
        }
        PORT mem_wr_n
        {
            direction = "output";
            width = "1";
            avalon_role = "writen";
        }
        PORT mem_rd_n
        {
            direction = "output";
            width = "1";
            avalon_role = "readn";
        }
        PORT reset_n
        {
            direction = "input";
            width = "1";
            avalon_role = "resetn";
        }
        PORT bus_wait
        {
            direction = "input";
            width = "1";
            avalon_role = "waitrequest";
        }
    }
}
MODULE boot_monitor_rom
{
    class = "altera_avalon_onchip_memory";
    class_version = "1.1";
    HDL_INFO
    {
        MIF_Files = "./boot_monitor_rom.mif";
        Synthesis_HDL_Files = "./boot_monitor_rom.v";
    }
    SYSTEM_BUILDER_INFO
    {
        Instantiate_In_System_Module = "1";
        Is_Enabled = "1";
        Is_Bus_Master = "0";
        Is_Memory_Device = "1";
        Uses_Tri_State_Data_Bus = "0";
        Address_Alignment = "dynamic";
        Address_Width = "8";
        Data_Width = "32";
        Has_IRQ = "0";
        Module_Desc = "memory";
```

```
        Base_Address = "0x0";
        Address_Span = "1024";
        Read_Wait_States = "0";
        Write_Wait_States = "0";
        Date_Modified = "2001.1.5.4:32:28";
        IRQ_Number = "N/A";
        Tri_State_Data_Bus = "";
    }
    WIZARD_SCRIPT_ARGUMENTS
    {
        Writeable = "0";
        Contents = "germs";
        Initfile = "";
    }
    PORT_WIRING
    {
        PORT address
        {
            direction = "input";
            width = "8";
            avalon_role = "address";
        }
        PORT read_data
        {
            direction = "output";
            width = "32";
            avalon_role = "readdata";
        }
    }
}
MODULE uart1
{
    class = "altera_avalon_uart";
    class_version = "1.1";
    HDL_INFO
    {
        Synthesis_HDL_Files = "./uart1.v";
    }
    SYSTEM_BUILDER_INFO
    {
        Instantiate_In_System_Module = "1";
        Is_Enabled = "1";
        Is_Bus_Master = "0";
        Is_Printable_Device = "1";
        Uses_Tri_State_Data_Bus = "0";
        Address_Alignment = "native";
        Address_Width = "3";
        Data_Width = "16";
```

```
    Has_IRQ = "1";
    IRQ_Number = "26";
    Module_Desc = "uart";
    Base_Address = "0x400";
    Read_Wait_States = "1";
    Write_Wait_States = "0";
    Date_Modified = "2001.1.5.5:54:51";
    Tri_State_Data_Bus = "";
}
WIZARD_SCRIPT_ARGUMENTS
{
    baud = "115200";
    data_bits = "8";
    fixed_baud = "1";
    parity = "N";
    stop_bits = "2";
    clock_freq = "33333000";
}
PORT_WIRING
{
    PORT txd
    {
        direction = "output";
        width = "1";
    }
    PORT data_from_cpu
    {
        direction = "input";
        width = "16";
        avalon_role = "writedata";
    }
    PORT rxd
    {
        direction = "input";
        width = "1";
    }
    PORT data_to_cpu
    {
        direction = "output";
        width = "16";
        avalon_role = "readdata";
    }
    PORT uart_select
    {
        direction = "input";
        width = "1";
        avalon_role = "chipselect";
    }
```

```
        PORT aggregate_irq
        {
           direction = "output";
           width = "1";
           avalon_role = "irq";
        }
        PORT clk
        {
           direction = "input";
           width = "1";
           avalon_role = "clk";
        }
        PORT reset_n
        {
           direction = "input";
           width = "1";
           avalon_role = "resetn";
        }
        PORT mem_r_wn
        {
           direction = "input";
           width = "1";
           avalon_role = "writen";
        }
        PORT mem_addr
        {
           direction = "input";
           width = "3";
           avalon_role = "address";
        }
    }
}
MODULE seven_seg_pio
{
    class = "altera_avalon_pio";
    class_version = "1.1";
    HDL_INFO
    {
       Synthesis_HDL_Files = "./seven_seg_pio.v";
    }
    SYSTEM_BUILDER_INFO
    {
       Date_Modified = "2001.0.26.11:42:39";
       Module_Desc = "pio";
       Is_Enabled = "1";
       Is_Bus_Master = "0";
       Instantiate_In_System_Module = "1";
       Uses_Tri_State_Data_Bus = "0";
```

```
    Has_IRQ = "0";
    IRQ_Number = "N/A";
    Address_Width = "2";
    Data_Width = "16";
    Address_Alignment = "native";
    Read_Wait_States = "1";
    Write_Wait_States = "0";
    Base_Address = "0x420";
    Tri_State_Data_Bus = "";
}
WIZARD_SCRIPT_ARGUMENTS
{
    has_tri = "0";
    has_out = "1";
    has_in = "0";
    edge_type = "NONE";
    irq_type = "NONE";
}
PORT_WIRING
{
    PORT mem_be_n
    {
        direction = "input";
        width = "2";
        avalon_role = "byteenablen";
    }
    PORT clk
    {
        direction = "input";
        width = "1";
        avalon_role = "clk";
    }
    PORT mem_addr
    {
        direction = "input";
        width = "2";
        avalon_role = "address";
    }
    PORT pio_select
    {
        direction = "input";
        width = "1";
        avalon_role = "chipselect";
    }
    PORT data_from_cpu
    {
        direction = "input";
        width = "16";
```

```
            avalon_role = "writedata";
        }
        PORT data_to_cpu
        {
            direction = "output";
            width = "16";
            avalon_role = "readdata";
        }
        PORT out_port
        {
            direction = "output";
            width = "16";
        }
        PORT reset_n
        {
            direction = "input";
            width = "1";
            avalon_role = "resetn";
        }
        PORT mem_r_wn
        {
            direction = "input";
            width = "1";
            avalon_role = "writen";
        }
    }
}
MODULE timer1
{
    class = "altera_avalon_timer";
    class_version = "1.1";
    HDL_INFO
    {
        Synthesis_HDL_Files = "./timer1.v";
    }
    SYSTEM_BUILDER_INFO
    {
        Instantiate_In_System_Module = "1";
        Is_Enabled = "1";
        Is_Bus_Master = "0";
        Uses_Tri_State_Data_Bus = "0";
        Address_Alignment = "native";
        Address_Width = "3";
        Data_Width = "16";
        Has_IRQ = "1";
        IRQ_Number = "25";
        Module_Desc = "timer";
        Base_Address = "0x440";
```

```
    Read_Wait_States = "1";
    Write_Wait_States = "0";
    Date_Modified = "2001.0.26.11:42:58";
    Tri_State_Data_Bus = "";
}
WIZARD_SCRIPT_ARGUMENTS
{
}
PORT_WIRING
{
    PORT irq
    {
        direction = "output";
        width = "1";
        avalon_role = "irq";
    }
    PORT data_from_cpu
    {
        direction = "input";
        width = "16";
        avalon_role = "writedata";
    }
    PORT data_to_cpu
    {
        direction = "output";
        width = "16";
        avalon_role = "readdata";
    }
    PORT clk
    {
        direction = "input";
        width = "1";
        avalon_role = "clk";
    }
    PORT reset_n
    {
        direction = "input";
        width = "1";
        avalon_role = "resetn";
    }
    PORT mem_r_wn
    {
        direction = "input";
        width = "1";
        avalon_role = "writen";
    }
    PORT mem_addr
    {
```

```
            direction = "input";
            width = "3";
            avalon_role = "address";
         }
         PORT timer_select
         {
            direction = "input";
            width = "1";
            avalon_role = "chipselect";
         }
      }
   }
   MODULE led_pio
   {
      class = "altera_avalon_pio";
      class_version = "1.1";
      HDL_INFO
      {
         Synthesis_HDL_Files = "./led_pio.v";
      }
      SYSTEM_BUILDER_INFO
      {
         Date_Modified = "2001.0.26.11:43:37";
         Module_Desc = "pio";
         Is_Enabled = "1";
         Is_Bus_Master = "0";
         Instantiate_In_System_Module = "1";
         Uses_Tri_State_Data_Bus = "0";
         Has_IRQ = "0";
         IRQ_Number = "N/A";
         Address_Width = "2";
         Data_Width = "2";
         Address_Alignment = "native";
         Read_Wait_States = "1";
         Write_Wait_States = "0";
         Base_Address = "0x460";
         Tri_State_Data_Bus = "";
      }
      WIZARD_SCRIPT_ARGUMENTS
      {
         has_tri = "1";
         has_out = "0";
         has_in = "0";
         edge_type = "NONE";
         irq_type = "NONE";
      }
      PORT_WIRING
      {
```

```
PORT mem_be_n
{
   direction = "input";
   width = "1";
   avalon_role = "byteenablen";
}
PORT bidir_port
{
   direction = "inout";
   width = "2";
}
PORT clk
{
   direction = "input";
   width = "1";
   avalon_role = "clk";
}
PORT mem_addr
{
   direction = "input";
   width = "2";
   avalon_role = "address";
}
PORT pio_select
{
   direction = "input";
   width = "1";
   avalon_role = "chipselect";
}
PORT data_from_cpu
{
   direction = "input";
   width = "2";
   avalon_role = "writedata";
}
PORT data_to_cpu
{
   direction = "output";
   width = "2";
   avalon_role = "readdata";
}
PORT reset_n
{
   direction = "input";
   width = "1";
   avalon_role = "resetn";
}
PORT mem_r_wn
```

```
        {
            direction = "input";
            width = "1";
            avalon_role = "writen";
        }
    }
}
MODULE button_pio
{
    class = "altera_avalon_pio";
    class_version = "1.1";
    HDL_INFO
    {
        Synthesis_HDL_Files = "./button_pio.v";
    }
    SYSTEM_BUILDER_INFO
    {
        Date_Modified = "2001.0.26.11:44:10";
        Module_Desc = "pio";
        Is_Enabled = "1";
        Is_Bus_Master = "0";
        Instantiate_In_System_Module = "1";
        Uses_Tri_State_Data_Bus = "0";
        Has_IRQ = "1";
        IRQ_Number = "27";
        Address_Width = "2";
        Data_Width = "12";
        Address_Alignment = "native";
        Read_Wait_States = "1";
        Write_Wait_States = "0";
        Base_Address = "0x470";
        Tri_State_Data_Bus = "";
    }
    WIZARD_SCRIPT_ARGUMENTS
    {
        has_tri = "0";
        has_out = "0";
        has_in = "1";
        edge_type = "ANY";
        irq_type = "EDGE";
    }
    PORT_WIRING
    {
        PORT mem_be_n
        {
            direction = "input";
            width = "2";
            avalon_role = "byteenablen";
```

```
}
PORT clk
{
   direction = "input";
   width = "1";
   avalon_role = "clk";
}
PORT mem_addr
{
   direction = "input";
   width = "2";
   avalon_role = "address";
}
PORT pio_select
{
   direction = "input";
   width = "1";
   avalon_role = "chipselect";
}
PORT in_port
{
   direction = "input";
   width = "12";
}
PORT irq
{
   direction = "output";
   width = "1";
   avalon_role = "irq";
}
PORT data_from_cpu
{
   direction = "input";
   width = "12";
   avalon_role = "writedata";
}
PORT data_to_cpu
{
   direction = "output";
   width = "12";
   avalon_role = "readdata";
}
PORT reset_n
{
   direction = "input";
   width = "1";
   avalon_role = "resetn";
}
```

```
        PORT mem_r_wn
        {
            direction = "input";
            width = "1";
            avalon_role = "writen";
        }
    }
}
MODULE lcd_pio
{
    class = "altera_avalon_pio";
    class_version = "1.1";
    HDL_INFO
    {
        Synthesis_HDL_Files = "./lcd_pio.v";
    }
    SYSTEM_BUILDER_INFO
    {
        Date_Modified = "2001.0.26.11:44:51";
        Module_Desc = "pio";
        Is_Enabled = "1";
        Is_Bus_Master = "0";
        Instantiate_In_System_Module = "1";
        Uses_Tri_State_Data_Bus = "0";
        Has_IRQ = "0";
        IRQ_Number = "N/A";
        Address_Width = "2";
        Data_Width = "11";
        Address_Alignment = "native";
        Read_Wait_States = "1";
        Write_Wait_States = "0";
        Base_Address = "0x480";
        Tri_State_Data_Bus = "";
    }
    WIZARD_SCRIPT_ARGUMENTS
    {
        has_tri = "1";
        has_out = "0";
        has_in = "0";
        edge_type = "NONE";
        irq_type = "NONE";
    }
    PORT_WIRING
    {
        PORT mem_be_n
        {
            direction = "input";
            width = "2";
```

```
    avalon_role = "byteenablen";
}
PORT bidir_port
{
    direction = "inout";
    width = "11";
}
PORT clk
{
    direction = "input";
    width = "1";
    avalon_role = "clk";
}
PORT mem_addr
{
    direction = "input";
    width = "2";
    avalon_role = "address";
}
PORT pio_select
{
    direction = "input";
    width = "1";
    avalon_role = "chipselect";
}
PORT data_from_cpu
{
    direction = "input";
    width = "11";
    avalon_role = "writedata";
}
PORT data_to_cpu
{
    direction = "output";
    width = "11";
    avalon_role = "readdata";
}
PORT reset_n
{
    direction = "input";
    width = "1";
    avalon_role = "resetn";
}
PORT mem_r_wn
{
    direction = "input";
    width = "1";
    avalon_role = "writen";
```

```
        }
    }
}
MODULE ext_ram
{
    class = "altera_nios_dev_board_sram32";
    class_version = "1.1";
    HDL_INFO
    {
    }
    PORT_WIRING
    {
        PORT data
        {
            width = "32";
            is_shared = "1";
            direction = "inout";
            avalon_role = "data";
        }
        PORT address
        {
            width = "16";
            is_shared = "1";
            direction = "input";
            avalon_role = "address";
        }
        PORT read_n
        {
            width = "1";
            is_shared = "1";
            direction = "input";
            avalon_role = "readn";
        }
        PORT write_n
        {
            width = "1";
            is_shared = "0";
            direction = "input";
            avalon_role = "writen";
        }
        PORT be_n
        {
            width = "4";
            is_shared = "1";
            direction = "input";
            avalon_role = "byteenablen";
        }
        PORT select0_n
```

```
        {
            width = "1";
            is_shared = "0";
            direction = "input";
            avalon_role = "registeredselectn";
        }
        PORT select1_n
        {
            width = "1";
            is_shared = "0";
            direction = "input";
            avalon_role = "registeredselectn";
        }
    }
    SYSTEM_BUILDER_INFO
    {
        Is_Enabled = "1";
        Instantiate_In_System_Module = "0";
        Is_Bus_Master = "0";
        Is_Memory_Device = "1";
        Uses_Tri_State_Data_Bus = "1";
        Uses_Registered_Select_Signal = "1";
        Address_Alignment = "dynamic";
        Data_Width = "32";
        Address_Width = "16";
        Has_IRQ = "0";
        IRQ_Number = "N/A";
        Read_Wait_States = "0";
        Write_Wait_States = "0";
        Hold_Time = "half_clock";
        Instance_Name = "--unknown--";
        Base_Address = "0x40000";
        Tri_State_Data_Bus = "ext_ram_bus";
        Date_Modified = "Jan_11_2001";
    }
}
MODULE ext_flash
{
    class = "altera_nios_dev_board_flash";
    class_version = "1.1";
    HDL_INFO
    {
    }
    PORT_WIRING
    {
        PORT data
        {
            width = "16";
```

```
        is_shared = "1";
        direction = "inout";
        avalon_role = "data";
    }
    PORT address
    {
        width = "19";
        is_shared = "1";
        direction = "input";
        avalon_role = "address";
    }
    PORT read_n
    {
        width = "1";
        is_shared = "1";
        direction = "input";
        avalon_role = "readn";
    }
    PORT write_n
    {
        width = "1";
        is_shared = "0";
        direction = "input";
        avalon_role = "writen";
    }
    PORT be_n
    {
        width = "2";
        is_shared = "1";
        direction = "input";
        avalon_role = "byteenablen";
    }
    PORT select_n
    {
        width = "1";
        is_shared = "0";
        direction = "input";
        avalon_role = "registeredselectn";
    }
}
SYSTEM_BUILDER_INFO
{
    Is_Enabled = "1";
    Instantiate_In_System_Module = "0";
    Is_Bus_Master = "0";
    Is_Memory_Device = "1";
    Uses_Tri_State_Data_Bus = "1";
    Uses_Registered_Select_Signal = "1";
```

```
            Address_Alignment = "dynamic";
            Data_Width = "16";
            Address_Width = "19";
            Has_IRQ = "0";
            IRQ_Number = "N/A";
            Read_Wait_States = "8";
            Write_Wait_States = "8";
            Instance_Name = "--unknown--";
            Base_Address = "0x100000";
            Tri_State_Data_Bus = "ext_ram_bus";
            Date_Modified = "Jan_10_2001";
        }
    }
    System_Wizard_Version = "1.1";
    System_Wizard_Build = "0";
}
```

****
********** END EXAMPLE **********
****